POCKET COMPUTER NEWSLETTER

© Copyright 1983 SCELBI Publications

A LOADER/MONITOR/DISASSEMBLER PROGRAM FOR THE SHARP PC-1500 AND RADIO SHACK PC-2

This combination loader/monitor/disassembler (LMD for the sake of brevity in the future) is a useful (essential!) aid to programming the Sharp PC-1500 or Radio Shack PC-2 pocket computers when using machine language.

The mnemonics used, with a few minor revisions, are those originally published in the 1982 Special Edition of *PCN*.

The LMD program is a "hybrid" in that it is written both in machine language (the majority of the code) and BASIC.

The loader might be considered as a cross between hand assembly and an assembler. Since source code must be written using hexadecimal opcodes (rather than mnemonics), it would be misleading to call this program an assembler. Naming it a loader seems more appropriate. The loader does have some things in common with an assembler. Labels may be used to mark both program locations and data storage locations. Furthermore, insertion and deletion of codes are easily accomplished. Assembly of the final source code takes only a fraction of a second.

An assembler that accepts mnemonics in source code is certainly a possibility for the PC-1500/PC-2. However, a true assembler would require three or four times as much memory as the loader program. This package represents a compromise that enables the LMD, a BASIC program, some source code and some object code to reside in memory at one time.

The following commands, executed by the indicated user-defined keys, are provided by the LMD program:

DEF C	**Convert to hexadecimal equivalent.**
DEF V	**View contents of memory.**
DEF H	**Hunt through memory for specified byte.**
DEF S	**Store code into memory.**
DEF D	**Disassemble code into mnemonics.**
DEF L	**List codes or mnemonics using printer.**
DEF J	**Jump to ML program.**
DEF M	**Move block of memory.**
DEF B	**Breakpoint set in user's ML program.**
DEF F	**Fix ML program following breakpoint error.**
DEF A	**Assemble source code into object code.**
DEF X	**Exit from View, Hunt, Store or Disassembler.**

Memory Utilization

A CE-155 8K RAM expansion module is required for use of the LMD program. The program is loaded from cassette by execution of CLOAD M. Load time is slightly under five minutes.

Memory is allocated as follows:

3800-38C4	**Reserve memory (197 bytes)**
38C5-3CFF	**User ML program area (1083 bytes)**
3D00-4839	**LMD system program & tables (2874 bytes)**
483A-4C3B	**LMD BASIC program (1026 bytes)**
4C3C-5FFF	**User source code & BASIC area (5060 bytes)**

Variables O, P, Q, R, O$, P$ and R$ are used by the LMD program and must be avoided in the user's BASIC program.

Note that if Reserve memory is not being used for some other purpose, the user ML program area may be expanded to include the area 3800-3CFF (a total of 1280 bytes).

Using the LMD

Each of the available commands is discussed in this publication.

Inputs made in response to prompts should be in hexadecimal, without the "&" prefix. Four hexadecimal digits are required for input of an address and two for data. Note that (for example) the code 0B should be entered by keying both 0 and B, not simply B.

ERROR 1 results when an incorrect entry is made. Excess inputs are ignored.

Conversion to Hexadecimal

Keying DEF C will convert a displayed expression (in decimal) to the hexadecimal equivalent in the range -FFFF to FFFF. (Out of range values result in the message: ERROR 1 IN 20.) Only the integer part of an input is used.

Examples:

(1) Enter 253, key DEF C, observe that FD, the hexadecimal equivalent of 253, is displayed.

Disassembly of Machine Language

DEF D initiates disassembly of ML programs on the LCD screen, with stored ML codes being displayed in mnemonic form. A prompt requests the beginning address. Keying ENTER causes the next mnemonic to be displayed. DEF X may be used to exit from the disassemble mode.

Example: Following DEF D, input 3D10 and ENTER. The display will show

3D10 INX

indicating that the code for the ML instruction to increment CPU register X is stored at that location. Key ENTER again and note that the next instruction is

3D11 LDAI (X)

In the case of relative branch instructions, the address branched to is computed and shown, separated from the instruction by a comma.

NOTE: The tape cassette on which this program is supplied is prepared by specially-designed digital high speed duplicating equipment. You may have to set volume and tone settings different than you normally use for tapes produced on your recorder. We appear to obtain best results, when using a Radio Shack Minisette-9 recorder, with the volume up in the range 7 to 9 and the tone down in the range 0 to 5. These settings may be different on your unit. However, regardless of the recorder being used, if you do not see the program title come up within about 10 seconds (immediately following the first burst of data), then the program will not load properly. Stop the unit, reposition the tape back to the beginning, and change the volume and/or tone controls to a new setting before trying again.

Example: Begin disassembly at 3D14. The display will show

3D14 FBNZ #33,3D49

indicating a branch (provided flag Z is not set) forward through memory 33 bytes to the instruction stored at address 3D49. (Remember that the number of bytes, 33, is in hexadecimal so that it is 51 bytes in decimal!)

The address of the routine called by a vector CALL instruction is also shown, along with the instruction, separated by a comma.

Example: Dissamble 3DC1 to obtain

3DC1 CALL #6C,F6FB

The address of the subroutine called, F6FB, was obtained from the table of vector calls located in ROM from FF00 through FFFF.

In certain ROM subroutines, one or more bytes (of data following the instruction calling that subroutine) serve as parameters "passed" to the subroutine. The disassembler will show these as passed bytes.

Example: Disassemble beginning at 4797 to obtain

4797 CALL CA, C001
PASS 65

Here the byte 65, located at address 4798, is passed to the subroutine beginning at C001. (Frequent use of passed bytes in the ROM may be observed by disassembling code beginning at address D817.)

Instructions addressing the second 64K bank of memory are indicated by an asterisk preceding the mnemonic for the instruction. A number of such instructions may be observed by starting disassembly at E550.

User ML programs that call subroutines in the PC's ROM should utilize vector CALLs rather than JSR instructions (where appropriate). This assures correct disassembly in cases where data is passed to a subroutine. (Example: Use CALL 0E rather than JSR D461.)

An attempt to disassemble codes that are not legal opcodes will produce the response: ERROR 1 IN 24.

Listing Memory Contents On A Printer

Key DEF L and respond to the prompts requesting the beginning and ending addresses of the codes to be listed. In response to the prompt "C OR M?", enter C to list machine codes (a "memory dump") or M to obtain mnemonics.

After a mnemonic listing has been completed, additional uses of the ENTER key will cause additional instructions to be printed. Use DEF X to discontinue operations.

Either CSIZE 1 or 2 may be used with the LIST function.

Examples:

(1) Key DEF L, input 3D14 and ENTER, then 3D1D and ENTER, finally L and ENTER. Observe that the machine codes in addresses 3D14 to 3D1D are printed.

(2) Key DEF L, input 3D14 and ENTER, then 3D1D and ENTER, finally M and ENTER. Observe that the mnemonics for the code in addresses 3D14 to 3D1D are printed.

The message "ERROR 1 IN 66" results when an attempt is made to print mnemonics for codes that are not legal ML directives.

Alternate Addressing

To use VIEW or LIST to show codes contained in the second 64K bank of memory, you must first store the code FD into address 3E60. Restore normal operation of the LMD by storing the code 38 into 3E60.

Example: Key DEF S and input 3E60 then ENTER. Input FD and ENTER. Key DEF L, input B000 then ENTER, input B00F and ENTER, input C and ENTER. The portion of the second 64K memory bank that is used by the CE-150 Printer/Cassette interface will be listed. Restore normal operation by keying DEF S, inputting 3E60 then ENTER and inputting 38 followed by ENTER.

VIEW, LIST and DISASSEMBLE may be used to study the contents of the ROM 2 area (used by the CE-158) by storing the following codes beginning at address 3E60.

For ROM 2a: Codes A8 E1 45
For ROM 2b: Codes A8 E3 45

To restore normal operation of the LMD, store the codes 38 05 44 beginning at address 3E60.

Example: With the CE-158 connected, store A8 E3 45 beginning at 3E60 using the DEF S option. With the CE-158 connected, key DEF L, input 8169 and ENTER, input 817E and ENTER, input M and ENTER. After the ROM listing has been completed, store 38 05 44 beginning at 3E60, using DEF S.

Jumping to a Machine Language Program

DEF J may be used, in lieu of a BASIC "CALL" statement, to begin execution of a ML program at a specific address.

Example: Using STORE, enter the following codes beginning at address 38C5

B5 55 AE 3A 00 9A

Use DISASSEMBLE, beginning at 38C5, to observe that the above codes constitute a simple ML program that stores the byte 55 into the address 3A00.

To execute this short ML program, key DEF J, input 38C5 and ENTER.

VIEW may then be used to verify that the byte 55 has been stored into the location 3A00.

Note that the ML program could also have been executed by using the BASIC instruction CALL &38C5.

Moving Blocks of Code

A block of stored codes may be moved in either direction. The new location may overlap the original block. The original block remains unchanged (unless overlapping occurs).

Beginning and ending addresses of the original block are requested by prompts once DEF M has been invoked. If the user inadvertently gives these addresses in the wrong order, the message "OUT OF ORDER" will be displayed.

Example: The short ML program (in the preceding example) stored in memory at 38C5-38CA can be relocated to addresses 3980-3985. Key DEF M, input 38C5 and ENTER, then 38CA and ENTER and finally 3980 and ENTER. VIEW or DISASSEMBLE may be used to verify that addresses 3980-3985 now contain the block of code that is in 38C5-38CA.

Caution should be exercised when using this capability to avoid a move of data into an area whose contents should not be disturbed!

Breakpoint Operation

The purpose of the "breakpoint" routine is to aid in debugging user ML programs. It will display the status of the CPU registers and flags when a specified location is reached in a ML program. As a further aid, the contents of floating-point register R0 is also shown. This feature is particularly valuable when debugging ML programs that utilize arithmetic routines (that can be accessed on the BASIC ROM).

The breakpoint feature is initiated by keying DEF B. Prompts then request the address at which ML program operation is to commence as well as the address at which the breakpoint is to be located. (Both of these addresses must be those of *valid ML opcodes!*)

The user's ML program will then be executed until the breakpoint location is reached. When that occurs, the contents of the CPU registers as they appeared at that point will be displayed. Information is provided in the following sequence:

(1) The contents of P (program counter)
(2) The contents of A (accumulator)
(3) The symbols of the flags that are *set*
(4) The contents of registers X, Y and U
(5) The contents of R0 (in decimal notation)

The display cycle continues to repeat when ENTER is keyed. Use DEF X to exit from the display procedure.

The breakpoint routine removes (and saves) three bytes from the user's program, at the specified breakpoint location. These three bytes are replaced by an instruction to jump to a routine that saves the contents of the CPU for display. The three bytes that were removed from the user's program are then automatically replaced when the breakpoint is reached.

In the event a user's ML program terminates with a RTS instruction *before* reaching the breakpoint, the three bytes

 Copyright 1983 POCKET COMPUTER NEWSLETTER All Rights Reserved

are replaced and the message "NOT REACHED" is displayed.

However, if the user's ML program terminates in some other way prior to reaching the breakpoint, then the removed bytes are *not* automatically replaced. In such cases, providing that a complete lockup has not occurred, keying DEF F will "fix" (repair) the program by replacing the three bytes. (Note that DEF F is *not* required when the display shows either the CPU contents or the "NOT REACHED" message.)

Examples:

(1) STORE the following codes into memory, starting at 38C5 -- **B5 00 48 12 4A 34 58 56 5A 78 68 9A 6A BC 9A**. LIST the mnemonics for addresses 38C5-38D3. Note that the ML program just entered loads A with 00, X with 1234, Y with 5678 and U with 9ABC.

To demonstrate the use of the breakpoint function: Key DEF B, input 38C5 and ENTER, then 38D3 and ENTER. This causes the ML program to begin executing at 38C5 and break at 38D3.

When the breakpoint is reached the display should show that P=38D3, A=00, and that flags C, I and Z are set. Press the ENTER key and observe that X, Y and U contain the expected values. Key ENTER again to observe the contents of R0 as 14533. (This number is of no particular significance in this example.) Key DEF X to terminate the display cycle.

(2) With the same program still in memory, again begin execution at 38C5, but this time have the breakpoint occur at 38D4.

Key DEF B, input 38C5 and ENTER then input 38D4 and ENTER. Observe that instead of having the CPU register contents displayed, the message "NOT REACHED" is shown. This is because an RTS instruction was executed prior to reaching the breakpoint.

Assembling Source Code

Source code for a ML program is written as a sequence of hexadecimal numbers, entered in numbered-line format in PRO mode, similar to lines of BASIC. Line numbers 100 through 999 should be used for this source code. Line 999 must be an END statement.

Each line number must be followed by a quote mark, after which the characters representing hexadecimal numbers follow. Closing quotes must *not* be used at the end of the line. Spaces may be used to separate codes to improve readability. However, a space may not be used between the two hexadecimal digits forming a byte.

Keying DEF A initiates the assembly of the source code into the object code that constitutes the ML program in its final form. A prompt requests the address at which the storage of object code is to begin. Following execution of the assembly routine, the ending address of the object code is displayed.

Example: The following short ML program clears string variables A$ to D$ by storing zero bytes into addresses 78C0 to 78FF. It may be used to demonstrate the assembly of source code into object code.

Mnemonics	Opcodes
LDXH #78	48 78
LDXL #C0	4A C0
LDA #00	B5 00
STAI (X)	41
CPXL #00	4E 00
RBNZ #05	99 05
RTS	9A

The source code for this program can be loaded by switching to the PRO mode and entering the lines:

```
100 "48 78
110 "4A C0
120 "B5 00
130 "41
140 "4E 00
150 "99 05
160 "9A
```

The source code may then be assembled into object code as follows: Set the PC to the RUN mode. Key DEF A. Input 38C5 and ENTER. The display will soon show 38D0 as the address of the last byte of the ML program.

To show that the program has been assembled, either the disassemble or list capabilities of LMD may be used. The program itself may be tested by first placing character strings of your choosing into string variables A$, B$, C$ and D$. Next execute the ML program using DEF J. Finally, check the contents of the string variables (A$ - D$) to see that they have been cleared.

Note that the source code used above could just as easily been entered in the format:

```
100 "48 78 4A C0
110 "B5 00 41 4E 00
120 "99 05 9A
```

or even more compactly as:

```
100 "4878AC0B500414E0099059A
```

At this point it might appear that the use of source code when entering a ML program offers no advantages whatever, particularly since instruction opcodes need to be used instead of mnemonics. However, the practical usefulness of this approach will soon be revealed.

Using Labels in Source Code

Single-character labels may be used within the source code listing to identify locations within a program. These labels must take the form L0, L1, . . . , L9, LB, . . . , LZ. (Other symbols such as L$ or L+ are also valid.) Each label in source code marks the location of the machine code that immediately follows it.

Branch and jump instructions in source code may be completed using the symbols *0, *1, etc., in place of the byte or bytes that would otherwise be used to specify relative or absolute addresses. The assembler routine will place the appropriate addresses marked by the corresponding labels into the object code! It will also calculate relative addresses when appropriate.

Example: An alternate way of writing the source code for the earlier illustration, using a label, is as follows:

```
100 "48 78 4A C0
110 "B5 00 L1 41
120 "4E 00 99 *1 9A
```

Note that in this case, with label 1 marking the beginning of a loop, it is not necessary for the user to calculate the size of the required relative branch. The above source code, with DEF A keyed, will produce exactly the same object code as the earlier example. Note that labels do not appear in the object code! They are used only by the assembly process.

The program example could easily be modified using this labeling technique by merely altering the source code as desired and reassembling. Suppose in the above example that line 110 is changed to:

```
110 "B5 41 L1 41
```

and the routine reconverted to object code using the assemble capabilities. Now execution of the resultant ML program would result in A$ through D$ being filled with the code 41 (representing the letter A in ASCII format).

We could insert an instruction to increment register A with each pass of the loop by changing line 120 to read:

```
120 "D0 4E 00 99 *1 9A
```

The change in source code is easy to make. As easy as any change to BASIC program lines. Invoking the assemble procedure again will result in the recalculation of the required relative branch to accomodate the inserted byte of code. In longer, more complex ML programs, this automatic calculation of addresses and relative branches can be especially helpful to the programmer.

Identification of Data Locations using Labels

When a label marks a data storage location, the label must be followed by a "dummy" byte so that the assemble routine will reserve a space for that storage location. This "dummy" byte may be any value. (The value 00 will be used in the accompanying example.)

The instructions whose mnemonics are listed below may be completed by *0, *1, etc., in place of the two-byte address that would ordinarily be used to complete the instruction. The assemble routine will place the appropriate addresses into the object code.

```
A1  SUBA nnnn     A9  ANDA nnnn     E9  AND  nnnn #nn
A3  ADDA nnnn     AB  ORA  nnnn     EB  OR   nnnn #nn
A5  LDA  nnnn     AD  EORA nnnn     ED  BIT  nnnn #nn
A7  CPA  nnnn     AE  STA  nnnn     EF  ADNC nnnn #nn
                  AF  BITA nnnn
```

The next routine illustrates the labeling of data locations.

```
Data locations defined:     L1 00
                            L2 00
                            L3 00
Program:                    LDA   #5A
                            STA   *1
                            LDA   #42
                            STA   *2
                            JSR   *M
                            RTS
Subroutine:                 LM
                            CLRC
                            ADDA  *1
                            SUBA  *2
                            STA   *3
                            RTS
```

This program could be entered as source code in this format:

```
100 "L1 00 L2 00 L3 00
110 "B5 5A AE *1 B5 42 AE *2
120 "BE *M 9A
130 "LM F9 A3 *1 A1 *2 AE *3 9A
```

A study of the resulting assembled object code should clarify the label usage in this example.

By the way, another benefit of using labels with source code is that the object code may be stored at any available location in memory. The correct addresses to be contained in the object code will automatically be determined by the LMD assembly operation.

Loader Error Messages

The routine that performs assembly of labeled object code is capable of detecting certain types of common errors. When it does, it displays a numbered error message. The exact location in the source listing at which the error occurred may be observed by using the scroll up key. The following error codes and their definitions can assist the operator:

ERROR 0 IN 15 — Indicates a reference has been made to an undefined label.

ERROR 1 IN 15 — Occurs when an illegal symbol is used in source code or the quote at the beginning of a line is missing.

ERROR 2 IN 15 — Results when a relative branch to a label spans more than the allowed 255 bytes or when it branches to a label in the wrong direction.

BASIC Programs

Programs may be written partly in BASIC and partly in ML if desired. If this is done, you should write the BASIC portion of the program using lines numbered 1000 and higher. Remember, the line

```
999 END
```

separates source code from a BASIC program. This END line is *absolutely essential* for correct operation of the loader routine as well as for the cassette operations that will be described shortly. (The line number for the END statement does not necessarily have to be 999, but it *must* separate the source code from any user BASIC programming.)

A user's BASIC program may be executed using a command such as RUN 1000. Alternately, a directive that refers to a BASIC label may be used.

Listing Source Code and BASIC Programs

A printed listing of source code that has been prepared for use by the loader (assembling) routine of the LMD may be obtained by giving the command:

```
LLIST 100,999
```

A listing of a BASIC program would be obtained by entering:

```
LLIST 1000,
```

Clearing User Memory

Execution of CALL &4835 will effectively clear any source code and BASIC program lines stored by the user.

BASIC Portion of the Loader/Monitor/Disassembler

```
10:"X"CLS :END
15:"A"GOSUB 50:
   CALL &3D00,0:
   PRINT "END: ";
   P$:END
20:"C"AREAD Q:
   CALL &3DB3:
   CURSOR 21:
   PRINT R$:END
21:"V"GOSUB 50
22:CALL &3DCF,0:
   PRINT P$:GOTO
   22
23:"D"GOSUB 50
24:CALL &3F20,0:
   PRINT P$;Q$:IF
   QFOR R=1TO Q:
   CALL &3E35,0:
   PRINT "PASS";R
   $:NEXT R
25:GOTO 24
26:"H"CLS :INPUT
   "BYTE SOUGHT:
   ";Q$:CALL &3DE
   8:GOSUB 50
27:CALL &3DDA,0:
   PRINT P$:GOTO
   27
30:"S"GOSUB 50
31:CLS :CALL &3E4
   1,0:WAIT 0:
   PRINT P$;:WAIT
   :INPUT ": ";Q$
   :CALL &3DED,0:
   GOTO 31
32:O=O+1:GOTO 31
33:"M"GOSUB 52:IF
   O>PPRINT "OUT
   OF ORDER":END
34:INPUT "MOVE TO
   : ";P$:CALL &3
   DFC:END
35:GOTO 34
40:"J"GOSUB 50:
   CALL &3E88,0:
   END
41:"B"GOSUB 50
42:INPUT "BRKPT:
   ";P$:GOTO 44
43:GOTO 42
44:CALL &3E8A,0:
   IF PEEK &76F0
   CALL &3EFF:
   PRINT "NOT REA
   CHED":END
45:PRINT Q$+P$:
   PRINT "X,Y,U:"
   ;R$:PRINT "R0=
   ";R:GOTO 45
46:"F"CALL &3EFF:
   END
50:CLS :INPUT "BE
   GIN: ";P$:CALL
   &3E64,0:RETURN
51:GOTO 50
52:GOSUB 50
53:INPUT "END: ";
   P$:CALL &3E64,
   P:RETURN
54:GOTO 53
60:"L"GOSUB 52
61:R$="":INPUT "C
    OR M? ";R$:
   CLS
62:IF R$="M"GOTO
   66
63:IF R$<>"C"GOTO
   61
64:CALL &3DCF,0:
   IF 0<=PLPRINT
   P$:GOTO 64
65:LPRINT LEFT$ (
   P$,3*(P-O)+19)
   :END
66:CALL &3F20,0:
   LPRINT P$+Q$;:
   IF QGOSUB 70
67:LPRINT :IF O>P
   PRINT
68:GOTO 66
70:FOR R=1TO Q:IF
   (3AND R)=1
   LPRINT :TAB 4
71:CALL &3E35,0:
   LPRINT R$;:
   NEXT R:RETURN
80:"SS"GOSUB 85:
   CALL &4796
81:"SB"GOSUB 85:
   CALL &47A4
82:"LS"GOSUB 85:
   CALL &47D1
83:"LB"GOSUB 85:
   CALL &47DD
85:R$="":INPUT "F
   ILE NAME? ";R$
86:RETURN
999:END
```

 Copyright 1983 POCKET COMPUTER NEWSLETTER All Rights Reserved

Cassette Operations

To make a recording of source code, input RUN "SS". A prompt will request the file name to be used. (The file name is optional.)

To record a BASIC program created by the user, input RUN "SB". Again, respond to the prompt for an optional file name. The BASIC program recorded using this procedure may be loaded later, when the LMD program is not in memory, by the usual CLOAD directive.

To record a machine language program (the object code),

Machine Language Portion of the Loader/Monitor/Disassembler

```
3D00 FD 88 FD 5A
3D04 48 7B 4A 10
3D08 6A EF CD BA
3D0C CC 69 44 44
3D10 44 45 B7 22
3D14 89 33 45 B7
3D18 20 9B 05 B7
3D1C 4C 8B 11 B7
3D20 2A 8B 1B B7
3D24 0D 9B 19 46
3D28 BE ED 95 81
3D2C 74 51 9E 1A
3D30 68 7A 6A D0
3D34 45 D9 FD EA
3D38 94 61 14 2E
3D3C 9E 28 44 56
3D40 55 54 B7 A0
3D44 91 30 54 9E
3D48 33 B7 F1 89
3D4C 54 FD 1A CC
3D50 69 44 44 44
3D54 45 B7 22 89
3D58 51 45 B7 20
3D5C 9B 05 B7 4C
3D60 8B 09 B7 2A
3D64 8B 08 B7 0D
3D68 9B 19 54 44
3D6C 9E 15 68 7A
3D70 6A D0 45 D9
3D74 FD EA BE DB
3D78 CA 89 03 24
3D7C 8B 20 56 55
3D80 B7 A0 83 20
3D84 FD C8 24 10
3D88 2A A4 90 28
3D8C FD 8A B7 90
3D90 83 04 B7 88
3D94 89 03 BE DA
3D98 A8 A4 8B 0A
3D9C 68 02 FD 5A
3DA0 E0 FD 5A E4
3DA4 A4 51 24 51
3DA8 9E 51 B7 F1
3DAC 99 0D FD 18
3DB0 46 8E 8E F2
3DB4 48 79 4A 80
3DB8 BE DC 20 58
3DBC 77 5A 70 48
3DC0 7A CD 6C F1
3DC4 D9 B3 1D 51
3DC8 D0 00 AC FD
3DCC 28 8E 74 BE
3DD0 3E 41 6A 03
3DD4 BE 3E 37 88
3DD8 05 9A A5 79
3DDC 86 F2 99 03
3DE0 46 BE 3D CF
3DE4 46 46 46 9A
3DE8 BE 3E 7C 8E
3DEC 02 FD 5A 48
3DF0 77 4A 60 BE
3DF4 ED 95 81 7F
3DF8 51 FD 18 9A
3DFC BE 3E 64 F4
3E00 79 75 FD 28
3E04 FB A5 79 7E
3E08 00 2A A5 79
3E0C 7D 80 28 84
3E10 96 81 0E 89
3E14 04 04 16 81
3E18 08 F5 88 03
3E1C FD 62 93 07
3E20 9A 24 FD CA
3E24 FD DA 84 A2
3E28 08 94 A2 18
3E2C 47 53 88 04
3E30 FD 62 93 08
3E34 9A 5A 70 B5
3E38 20 58 77 51
3E3C BE 3E 60 8E
3E40 07 5A 50 84
3E44 BE 3E 48 04
3E48 58 77 28 F1
3E4C BE 3E 50 A4
3E50 F9 B9 0F B3
3E54 30 B7 3A 81
3E58 02 B3 06 51
3E5C 59 00 FB 9A
3E60 38 05 44 9A
3E64 4A 50 48 77
3E68 BE ED 95 81
3E6C 0A 18 BE ED
3E70 95 81 04 1A
3E74 FD 18 9A FD
3E78 1A FD 1A E4
3E7C 58 79 5A 84
3E80 B5 B2 51 B5
3E84 00 51 1E 9A
3E88 FD 5E FD 88
3E8C BE 3E 64 58
3E90 76 5A F0 5B
3E94 FF 54 84 51
3E98 04 51 F5 F5
3E9C F5 46 B5 A8
3EA0 43 B5 3E 43
3EA4 B5 BA 0E 9A
3EA8 AA 78 48 FD
3EAC A8 FD 98 FD
3EB0 88 FD C8 FD
3EB4 AA 48 3F 4A
3EB8 12 58 77 5A
3EBC 50 6A 06 F5
3EC0 88 03 6A 04
3EC4 D5 81 02 F5
3EC8 46 44 88 08
3ECC 59 00 5A 60
3ED0 F5 F5 48 76
3ED4 4A F1 BE 3E
3ED8 3C BE 3E 3C
3EDC B5 20 51 B5
3EE0 41 51 B5 3D
3EE4 51 FD 8A BE
3EE8 3E 48 5A 70
3EEC 6A 02 B5 20
3EF0 51 FD 0A BE
3EF4 3E 43 88 0A
3EF8 48 79 4A 88
3EFC BE DC 0C 48
3F00 76 4A F0 05
3F04 DD 89 0A 49
3F08 00 44 45 18
3F0C 45 1A F5 F5
3F10 F5 9A 20 46
3F14 4C 41 47 53
3F18 20 43 49 5A
3F1C 56 48 50 3D
3F20 BE 3E 41 B5
3F24 20 51 59 00
3F28 BE 3E 7C BE
3F2C 3E 60 28 58
3F30 77 5A 60 B7
3F34 FD 89 52 BE
3F38 3E 60 28 B7
3F3C 8C 8B 38 B7
3F40 9C 8B 34 B7
3F44 D3 8B 30 B7
3F48 D7 8B 2C B9
3F4C F0 B7 A0 8B
3F50 04 B7 30 83
3F54 0C A4 B9 0F
3F58 B7 0B 83 1B
3F5C D5 83 18 8E
3F60 1C B7 E0 8B
3F64 08 B7 40 81
3F68 14 B7 70 83
3F6C 10 A4 B9 0F
3F70 B7 09 81 09
3F74 D5 81 06 EB
3F78 77 54 2A 8E
3F7C 28 FD 88 48
3F80 46 4A 67 BE
3F84 40 B2 FD 0A
3F88 9A B7 C0 81
3F8C 18 D5 83 15
3F90 B5 43 51 B5
3F94 41 51 B5 4C
3F98 51 51 46 BE
3F9C 3E 37 B5 2C
3FA0 51 B5 20 8E
3FA4 A9 FD 88 48
3FA8 40 4A DD BE
3FAC 40 B2 FD 0A
3FB0 B7 01 81 E1
3FB4 8B 1C B7 06
3FB8 8B 3A 83 8D
3FBC B7 04 8B 17
3FC0 83 1D B7 03
3FC4 8B 06 BE 3E
3FC8 3C BA 3E 3C
3FCC BE 3F C6 B5
3FD0 20 51 B5 23
3FD4 BA 3E 3B BE
3FD8 3F D2 FD 88
3FDC A4 8E 0A BE
3FE0 3F D2 FD 88
3FE4 B5 00 A0 FD
3FE8 42 FD CA B5
3FEC 2C 51 BE 3E
3FF0 43 FD 0A 9A
3FF4 BE 3E 3C A4
3FF8 2A BE 3E 3C
3FFC 24 B7 CC 89
4000 04 6C C0 BE
4004 3E B7 CF 89
4008 08 6C 27 89
400C 88 B5 04 8E
4010 36 B7 D1 8B
4014 30 B7 D2 89
4018 04 6C E0 83
401C 28 B7 D4 81
4020 24 B7 D8 81
4024 20 B7 DA 89
4028 04 6C B2 83
402C 18 B7 DB 89
4030 04 6C BB 81
4034 10 B7 DD 8B
4038 0C B7 DF 89
403C 58 6C 9B 81
4040 54 6C B2 83
4044 50 B5 01 8E
4048 49 BE 3F D2
404C B5 2C 51 FD
4050 88 A4 2A 68
4054 FF 65 08 67
4058 0A BE 3E 43
405C FD 0A BE 3E
4060 60 46 28 24
4064 D5 B7 18 81
4068 06 B7 63 81
406C 11 B1 4A 58
4070 40 5A 97 D5
4074 FD DA 15 83
4078 01 F1 B9 0F
407C 8E 14 B7 1A
4080 89 06 A4 D9
4084 B3 03 8E 0A
4088 B7 61 B5 00
408C 81 04 6C E0
4090 B3 02 AE 79
4094 86 FB 9A 33
4098 10 10 02 10
409C 00 01 10 00
40A0 01 12 11 00
40A4 11 12 22 11
40A8 00 01 00 00
40AC 00 00 00 22
40B0 00 00 45 B7
40B4 FF 8B 20 A6
40B8 8B 07 B5 10
40BC F7 91 03 9E
40C0 0F 45 51 B7
40C4 20 93 06 2A
40C8 B5 20 56 51
40CC 5E 05 83 03
40D0 51 9E 07 59
40D4 00 24 9A AA
40D8 78 4A FD 1A
40DC E4 00 53 55
40E0 42 41 20 58
40E4 4C 00 01 53
40E8 55 42 41 20
40EC 28 58 29 00
40F0 02 41 44 44
40F4 41 20 58 4C
40F8 00 03 41 44
40FC 44 41 20 28
4100 58 29 00 04
4104 4C 44 41 20
4108 20 58 4C 00
410C 05 4C 44 41
4110 20 20 28 58
4114 29 00 06 43
4118 50 41 20 20
411C 58 4C 00 07
4120 43 50 41 20
4124 20 28 58 29
4128 00 08 53 54
412C 41 20 20 58
4130 48 00 09 41
4134 4E 44 41 20
4138 28 58 29 00
413C 0A 53 54 41
4140 20 20 58 4C
4144 00 0B 4F 52
4148 41 20 20 28
414C 58 29 00 0C
4150 44 53 42 41
4154 20 28 58 29
4158 00 0D 45 4F
415C 52 41 20 28
4160 58 29 00 0E
4164 53 54 41 20
4168 20 28 58 29
416C 00 0F 42 49
4170 54 41 20 28
4174 58 29 00 10
4178 53 55 42 41
417C 20 59 4C 00
4180 11 53 55 42
4184 41 20 28 59
4188 29 00 12 41
418C 44 44 41 20
4190 59 4C 00 13
4194 41 44 44 41
4198 20 28 59 29
419C 00 14 4C 44
41A0 41 20 20 59
41A4 4C 00 15 4C
41A8 44 41 20 20
41AC 28 59 29 00
41B0 16 43 50 41
41B4 20 20 59 4C
41B8 00 17 43 50
41BC 41 20 20 28
41C0 59 29 00 18
41C4 53 54 41 20
41C8 20 59 48 00
41CC 19 41 4E 44
41D0 41 20 28 59
41D4 29 00 1A 53
41D8 54 41 20 20
41DC 59 4C 00 1B
41E0 4F 52 41 20
41E4 20 28 59 29
41E8 00 1C 44 53
41EC 42 41 20 28
41F0 59 29 00 1D
41F4 45 4F 52 41
41F8 20 28 59 29
41FC 00 1E 53 54
4200 41 20 20 28
4204 59 29 00 1F
4208 42 49 54 41
420C 20 28 59 29
4210 00 20 53 55
4214 42 41 20 55
4218 4C 00 21 53
421C 55 42 41 20
4220 28 55 29 00
4224 22 41 44 44
4228 41 20 55 4C
422C 00 23 41 44
4230 44 41 20 28
4234 55 29 00 24
4238 4C 44 41 20
423C 20 55 4C 00
4240 25 4C 44 41
4244 20 20 28 55
4248 29 00 26 43
424C 50 41 20 20
4250 55 4C 00 27
4254 43 50 41 20
4258 20 28 55 29
425C 00 28 53 54
4260 41 20 20 55
4264 48 00 29 41
4268 4E 44 41 20
426C 28 55 29 00
4270 2A 53 54 41
4274 20 20 55 4C
4278 00 2B 4F 52
427C 41 20 20 28
4280 55 29 00 2C
4284 44 53 42 41
4288 20 28 55 29
428C 00 2D 45 4F
4290 52 41 20 28
4294 55 29 00 2E
4298 53 54 41 20
429C 20 28 55 29
42A0 00 2F 42 49
42A4 54 41 20 28
42A8 55 29 00 30
42AC 4E 4F 50 00
42B0 40 49 4E 58
42B4 4C 00 41 53
42B8 54 41 49 20
42BC 28 58 29 00
42C0 42 44 45 58
42C4 4C 00 43 53
42C8 54 41 44 20
42CC 28 58 29 00
42D0 44 49 4E 58
42D4 00 45 4C 44
42D8 41 49 20 28
42DC 58 29 00 46
42E0 44 45 58 00
42E4 47 4C 44 41
42E8 44 20 28 58
42EC 29 00 48 4C
42F0 44 58 48 01
42F4 49 41 4E 44
42F8 20 20 28 58
42FC 29 01 4A 4C
4300 44 58 4C 01
4304 4B 4F 52 20
4308 20 20 28 58
430C 29 01 4C 43
4310 50 58 48 01
4314 4D 42 49 54
4318 20 20 28 58
431C 29 01 4E 43
4320 50 58 4C 01
4324 4F 41 44 4E
4328 43 20 28 58
432C 29 01 50 49
4330 4E 59 4C 00
4334 51 53 54 41
4338 49 20 28 59
433C 29 00 52 44
4340 45 59 4C 00
4344 53 53 54 41
4348 44 20 28 59
434C 29 00 54 49
4350 4E 59 00 55
4354 4C 44 41 49
4358 20 28 59 29
435C 00 56 44 45
4360 59 00 57 4C
4364 44 41 44 20
4368 28 59 29 00
436C 58 4C 44 59
4370 48 01 59 41
4374 4E 44 20 20
4378 28 59 29 01
437C 5A 4C 44 59
4380 4C 01 5B 4F
4384 52 20 20 20
4388 28 59 29 01
438C 5C 43 50 59
4390 48 01 5D 42
4394 49 54 20 20
4398 28 59 29 01
439C 5E 43 50 59
43A0 4C 01 5F 41
43A4 44 4E 43 20
43A8 28 59 29 01
43AC 60 49 4E 55
43B0 4C 00 61 53
43B4 54 41 49 20
43B8 28 55 29 00
43BC 62 44 45 55
43C0 4C 00 63 53
43C4 54 41 44 20
43C8 28 55 29 00
43CC 64 49 4E 55
43D0 00 65 4C 44
43D4 41 49 20 28
43D8 55 29 00 66
43DC 44 45 55 00
43E0 67 4C 44 41
43E4 44 20 28 55
43E8 29 00 68 4C
43EC 44 55 48 01
43F0 69 41 4E 44
43F4 20 20 28 55
43F8 29 01 6A 4C
43FC 44 55 4C 01
4400 6B 4F 52 20
4404 20 20 28 55
4408 29 01 6C 43
440C 50 55 48 01
4410 6D 42 49 54
4414 20 20 28 55
4418 29 01 6E 43
441C 50 55 4C 01
4420 6F 41 44 4E
4424 43 20 28 55
4428 29 01 80 53
442C 55 42 41 20
4430 58 48 00 81
4434 46 42 4E 43
4438 04 82 41 44
443C 44 41 20 58
4440 48 00 83 46
4444 42 43 20 04
4448 84 4C 44 48
444C 20 20 58 48
4450 00 85 46 42
4454 4E 48 04 86
4458 43 50 41 20
445C 20 58 48 00
4460 87 46 42 48
4464 20 04 88 42
4468 4E 5A 44 05
446C 89 46 42 4E
4470 5A 04 8A 52
4474 54 49 00 8B
4478 46 42 5A 20
447C 04 8C 44 41
4480 44 41 20 28
4484 58 29 00 8D
4488 46 42 4E 56
448C 04 8E 46 42
4490 20 20 04 8F
4494 46 42 56 20
4498 04 90 53 55
449C 42 41 20 59
44A0 48 00 91 52
44A4 42 4E 43 05
44A8 92 41 44 44
44AC 41 20 59 48
44B0 00 93 52 42
44B4 43 20 05 94
44B8 4C 44 41 20
44BC 20 59 48 00
44C0 95 52 42 4E
44C4 48 05 96 43
44C8 50 41 20 20
44CC 59 48 00 97
44D0 52 42 48 20
44D4 05 99 52 42
44D8 4E 5A 05 9A
44DC 52 54 53 00
44E0 9B 52 42 5A
44E4 20 05 9C 44
44E8 41 44 41 20
44EC 28 59 29 00
44F0 9D 52 42 4E
44F4 56 05 9E 52
44F8 42 20 20 05
44FC 9F 52 42 56
4500 20 05 A0 53
4504 55 42 41 20
4508 55 48 00 A1
450C 53 55 42 41
4510 02 A2 41 44
4514 44 41 20 55
4518 48 00 A3 41
451C 44 44 41 02
4520 A4 4C 44 41
4524 20 20 55 48
4528 00 A5 4C 44
452C 41 20 02 A6
4530 43 50 41 20
4534 20 55 48 00
4538 A7 43 50 41
453C 20 02 A8 52
4540 4F 4D 32 00
4544 A9 41 4E 44
4548 41 02 AA 4C
454C 44 53 23 02
4550 AB 4F 52 41
4554 20 02 AC 44
4558 41 44 41 20
455C 28 55 29 00
4560 AD 45 4F 52
4564 41 02 AE 53
4568 54 41 20 02
456C AF 42 49 54
4570 41 02 B1 53
4574 55 42 41 01
4578 B3 41 44 44
457C 41 01 B5 4C
4580 44 41 20 01
4584 B7 43 50 41
4588 20 01 B8 52
458C 4F 4D 31 00
4590 B9 41 4E 44
4594 41 01 BA 4A
4598 4D 50 20 02
459C BB 4F 52 41
45A0 20 01 BD 45
45A4 4F 52 41 01
45A8 BE 4A 53 52
45AC 20 06 BF 42
45B0 49 54 41 01
45B4 C1 43 41 4E
45B8 43 07 C3 43
45BC 41 43 20 07
45C0 C5 43 41 4E
45C4 48 07 C7 43
45C8 41 48 20 07
45CC C9 43 41 4E
45D0 5A 07 CB 43
45D4 41 5A 20 07
45D8 CD 43 41 4C
45DC 4C 07 CF 43
45E0 41 56 20 07
45E4 D1 52 52 43
45E8 41 00 D3 52
45EC 44 52 20 20
45F0 28 58 29 00
45F4 D5 53 52 41
45F8 00 D7 52 44
45FC 4C 20 20 28
4600 58 29 00 D9
4604 53 4C 41 00
4608 DB 52 4C 43
460C 41 00 DD 49
4610 4E 41 00 DF
4614 44 45 41 00
4618 E1 52 4F 4D
461C 32 61 00 E3
4620 52 4F 4D 32
4624 62 00 E9 41
4628 4E 44 20 03
462C EB 4F 52 20
4630 20 03 ED 42
4634 49 54 20 03
4638 EF 41 44 4E
463C 43 03 F1 52
4640 44 41 00 F5
4644 53 54 49 20
4648 20 28 58 29
464C 28 59 29 00
4650 F7 43 50 41
4654 49 20 28 58
4658 29 00 F9 43
465C 4C 52 43 00
4660 FB 53 45 54
4664 43 00 FF 0A
4668 50 4F 50 58
466C 00 18 4C 44
4670 58 20 20 59
4674 00 1A 50 4F
4678 50 59 00 28
467C 4C 44 58 20
4680 20 55 00 2A
4684 50 4F 50 55
4688 00 40 49 4E
468C 58 48 00 42
4690 44 45 58 48
4694 00 48 4C 44
4698 58 20 20 53
469C 00 4C 50 57
46A0 52 20 4F 46
46A4 46 00 4E 53
46A8 54 58 20 20
46AC 53 00 50 49
46B0 4E 59 48 00
46B4 52 44 45 59
46B8 48 00 58 4C
46BC 44 58 20 20
46C0 50 00 5A 53
46C4 54 58 20 20
46C8 59 00 5E 53
46CC 54 58 20 20
46D0 50 00 60 49
46D4 4E 55 48 00
46D8 62 44 45 55
46DC 48 00 6A 53
46E0 54 58 20 20
46E4 55 00 81 53
46E8 45 54 49 00
46EC 88 50 53 48
46F0 58 00 8A 50
46F4 4F 50 41 00
46F8 8E 43 4C 52
46FC 44 00 98 50
4700 53 48 59 00
4704 A8 50 53 48
4708 55 00 AA 4C
470C 44 41 20 20
4710 45 00 B1 57
4714 41 49 00 BA
4718 4C 44 41 20
471C 20 4B 42 00
4720 BE 43 4C 52
4724 49 00 C0 44
4728 53 50 20 4F
472C 46 46 00 C1
4730 44 53 50 20
4734 4F 4E 00 C8
4738 50 53 48 41
473C 00 CA 41 44
4740 44 58 20 41
4744 00 CE 53 54
4748 30 41 20 54
474C 49 00 DA 41
4750 44 44 59 20
4754 41 00 DE 53
4758 54 31 41 20
475C 54 49 00 EA
4760 41 44 44 55
4764 20 41 00 EC
4768 53 54 41 20
476C 20 45 00 FF
4770 CC 69 44 44
4774 44 47 1A 45
4778 FD CA 5E 22
477C 9B 0C 9A B5
4780 00 AE 78 79
4784 24 BE BB D6
4788 44 58 27 5A
478C 70 6A 0F 55
4790 8B 03 41 88
4794 06 9A 6A 0B
4798 BE 47 2F BE
479C 47 70 FD 88
47A0 CC 69 8E 0C
47A4 6A 01 BE 47
47A8 7F CC 67 FD
47AC 88 BE 47 70
47B0 FD 2A FD 88
47B4 BE DF E1 81
47B8 16 64 F6 7B
47BC 84 FD 2A FD
47C0 AB F6 7B 82
47C4 CD B0 F4 7B
47C8 84 FD 0A CD
47CC AA CD B4 CD
47D0 42 CC 69 6A
47D4 08 BE 47 F4
47D8 BE 48 14 8E
47DC 10 BE 47 70
47E0 FD 88 6A 01
47E4 BE 47 F4 FD
47E8 0A CA 67 4B
47EC FF FD 1A FD
47F0 1A 6B 2C E0
47F4 FD 88 B5 80
47F8 BE 47 81 CD
47FC B0 FD 0A 83
4800 10 F4 7B AC
4804 CD AA 83 09
4808 46 CA 67 4B
480C FF CD B4 CD
4810 42 CD B4 9A
4814 48 48 4A 3A
4818 CA 65 48 4C
481C 4A 35 CA 69
4820 58 48 5A 2E
4824 6A 06 55 41
4828 88 04 46 CA
482C 67 9A 03 E7
4830 03 F1 BE 0D
4834 FF BE 48 14
4838 CD 42
```

Copyright 1983 POCKET COMPUTER NEWSLETTER All Rights Reserved

execute CSAVE M "File Name"; G(start address), G(ending address). The machine language program may be reloaded into its original memory locations using CLOAD M "File Name". (The use of a file name is also optional in this instance.)

When the LMD program is in memory, a previously saved source code listing may be loaded by execution of RUN "LS". If this is done, any source code or user BASIC lines presently in memory will be destroyed.

A previously saved user BASIC program may be loaded (while the LMD package is in memory) by execution of RUN "LB". Any user BASIC lines currently in memory will be lost, but source code lines will not be disturbed.

A backup copy of the LMD program may be made by executing CSAVE M "LMD"; &3D00, &4C34, &4835.

A SYNOPSIS OF PC-1500/PC-2 MACHINE LANGUAGE USING ROBER MNEMONICS

CPU Registers

The registers in the CPU can be described as follows:

A: Accumulator (8 bits)
X: Data pointer (16 bits). XH and XL are each 8-bit registers which may be combined to form the 16-bit X register.
Y: Data pointer (16 bits). Made up of YH and YL.
U: Data pointer (16 bits). Made up of UH and UL.
E: Flag register (8 bits, only 5 bits used.) Bit 0 is flag C (carry), bit 1 is flag I (interrupt), bit 2 is flag Z (zero). bit 3 is flag V (overflow) and bit 4 is flag H (half-carry).
P: Program counter (16 bits). Contains the address of the next instruction to be executed.
S: Stack pointer (16 bits). Contains the address of the next available stack position.

Memory Addressing

Two 64K memory banks are addressable. The first may contain either instructions or data. The second bank may only hold data. Most of the ML instructions addressing memory can be prefixed by the code FD. This causes the second bank to be addressed. A mnemonic preceded by an asterisk is used to denote such an instruction. The second 64K bank is used by the PC to address I/O ports.

The ROM area, 8000 through BFFF, may contain two ROMs at the same time. The CPU instructions ROM1 and ROM2 (opcodes B8 and A8) determine which of the two will be addressed. (ROM 1 is used by the CE-150 and ROM 2 by the CE-158.) ROM 2 is further split into two banks, ROM 2a and ROM 2 b, in the area 8000 through 9FFF. Opcodes E1 and E3 determine which sub-bank is selected.

Flags

Flag C is set to 1 when a carry results from an addition or increment or when a borrow does *not* result from a subtraction, decrement or comparison. It is cleared to 0 in the opposite cases. The C flag is also affected by some bit shift and rotate instructions.

Flag I enables interrupts when set, disables them when it is cleared.

Flag Z, when affected by a ML instruction, is set to 1 when the result of a load or operation is zero. Otherwise this flag is cleared to the 0 state.

Flag V is set to 1 when an overflow occurs (assuming that signed binary arithmetic is being performed) and cleared when no such overflow occurs. This flag may also be affected by bit rotate and shifting instructions.

Flag H is set to 1 by a carry (or a *lack* of a borrow) from the least significant four bits of a register to the most significant four bits (between hexadecimal digits). It is also affected by bit rotate and shift instructions.

Location of Data

Data involved in ML instructions is obtained from a CPU register, a memory address or from the instruction itself. A memory address may be specified directly as four hexadecimal digits. Or, it may be the address pointed to by the contents of X, Y or U, in which case it is denoted by the data pointer being enclosed in parenthesis: (X), (Y) or (U). Data that is part of an instruction is known as *immediate data* and is denoted by preceding the data byte with a pound (#) sign.

The Instruction Set

Load instructions, using the mnemonic LD, include specification of (1) the CPU register to be loaded, and (2) the register, memory address or immediate data to be loaded into that register. If the register loaded is the accumulator (A register) then flag Z is affected, otherwise it is not affected.

Store instructions have the mnemonic ST. They take the contents of the first register named and store it into the second register or indicated memory address. Flags are not affected by these directives.

Add instructions calculate the sum of the specified data bytes. The result is placed in the first-named register (or the indicated memory address). There are several variations:

(1) Addition to the accumulator includes the contents of the carry flag. The result of the addition affects flags C, Z, V and H.
(2) Decimal addition (performed using binary-coded decimal notation) is signified by the mnemonic DAD. The contents of flag C are included in the calculation. Flags C, Z, V and H are affected by the results of the operation.
(3) Addition without the carry is indicated by the mnemonic ADNC. The sum does *not* include the initial contents of the C flag in this case. However, flags C, Z, V and H are affected by the result.
(4) Addition to a 16-bit register completely ignores all flags. The mnemonics for this group are: ADDX, ADDY and ADDU.

Subtraction instructions use the mnemonic SUB. The second data byte specified is subtracted from the first. The result is placed into the first-named register. If the C flag is cleared (prior to the operation), 1 is added to the subtrahend. The mnemonic DSB is used to indicate decimal subtraction. Flags C, Z, V and H are affected by subtraction directives.

Increments and decrements of CPU registers are denoted by the mnemonics IN and DE and the name of the affected register. The beginning contents of the carry flag (C) are ignored. The increment or decrement of an 8-bit register affects the flags in the same manner as they would be if the value 1 was added or subtracted. However, incrementing or decrementing a 16-bit register does not affect the status of the flags.

In certain instructions increments or decrements of 16-bit registers are combined with loads and stores. For example, the instruction LDAI (X) performs a load with an increment. That is, register A is loaded with a data byte from address (X), after which the address pointer in register X is incremented.

The Boolean logic operations AND, OR and EOR (exclusive-OR) are performed between two specified bytes with the result being placed in the first-named register or memory location. Only the zero (Z) flag is affected by these directives.

Comparisons (signified by the mnemonic CP) operate similarly to subtractions except that (1) the previous contents of flag C are ignored, and (2) the result of the subtraction is not saved. Only the flags are affected by these operations.

Bit tests (having the mnemonic BIT) perform a logical AND and affect only the Z flag.

Two special instructions are worthy of particular note. One of them is useful when moving blocks of data. The mnemonic for this directive is STI (X)(Y). Its effect is to

 Copyright 1983 POCKET COMPUTER NEWSLETTER All Rights Reserved

Regular Opcodes and Mnemonics

Code	Mnemonic
B3	ADDA #nn
23	ADDA (U)
03	ADDA (X)
13	ADDA (Y)
A3	ADDA nnnn
A2	ADDA UH
22	ADDA UL
82	ADDA XH
02	ADDA XL
92	ADDA YH
12	ADDA YL
6F	ADNC (U) #nn
4F	ADNC (X) #nn
5F	ADNC (Y) #nn
EF	ADNC nnnn #nn
69	AND (U) #nn
49	AND (X) #nn
59	AND (Y) #nn
E9	AND nnnn #nn
B9	ANDA #nn
29	ANDA (U)
09	ANDA (X)
19	ANDA (Y)
A9	ANDA nnnn
6D	BIT (U) #nn
4D	BIT (X) #nn
5D	BIT (Y) #nn
ED	BIT nnnn #nn
BF	BITA #nn
2F	BITA (U)
0F	BITA (X)
1F	BITA (Y)
AF	BITA nnnn
88	BNZD #nn
C3	CAC #nn
C7	CAH #nn
CD	CALL #nn
C0	CALL C0
C2	CALL C2
C4	CALL C4
C6	CALL C6
C8	CALL C8
CA	CALL CA
CC	CALL CC
CE	CALL CE
D0	CALL D0
D2	CALL D2
D4	CALL D4
D6	CALL D6
D8	CALL D8
DA	CALL DA
DC	CALL DC
DE	CALL DE
E0	CALL E0
E2	CALL E2
E4	CALL E4
E6	CALL E6
E8	CALL E8
EA	CALL EA
EC	CALL EC
EE	CALL EE

Code	Mnemonic
F0	CALL F0
F2	CALL F2
F4	CALL F4
F6	CALL F6
C1	CANC #nn
C5	CANH #nn
C9	CANZ #nn
CF	CAV #nn
CB	CAZ #nn
F9	CLRC
B7	CPA #nn
27	CPA (U)
07	CPA (X)
17	CPA (Y)
A7	CPA nnnn
A6	CPA UH
26	CPA UL
86	CPA XH
06	CPA XL
96	CPA YH
16	CPA YL
F7	CPAI (X)
6C	CPUH #nn
6E	CPUL #nn
4C	CPXH #nn
4E	CPXL #nn
5C	CPYH #nn
5E	CPYL #nn
AC	DADA (U)
8C	DADA (X)
9C	DADA (Y)
DF	DEA
66	DEU
62	DEUL
46	DEX
42	DEXL
56	DEY
52	DEYL
2C	DSBA (U)
0C	DSBA (X)
1C	DSBA (Y)
BD	EORA #nn
2D	EORA (U)
0D	EORA (X)
1D	EORA (Y)
AD	EORA nnnn
8E	FB #nn
83	FBC #nn
87	FBH #nn
81	FBNC #nn
85	FBNH #nn
8D	FBNV #nn
89	FBNZ #nn
8F	FBV #nn
8B	FBZ #nn
DD	INA
64	INU
60	INUL
44	INX
40	INXL
54	INY

Code	Mnemonic
50	INYL
BA	JMP nnnn
BE	JSR nnnn
B5	LDA #nn
25	LDA (U)
05	LDA (X)
15	LDA (Y)
A5	LDA nnnn
A4	LDA UH
24	LDA UL
84	LDA XH
04	LDA XL
94	LDA YH
14	LDA YL
67	LDAD (U)
47	LDAD (X)
57	LDAD (Y)
65	LDAI (U)
45	LDAI (X)
55	LDAI (Y)
AA	LDS# nnnn
68	LDUH #nn
6A	LDUL #nn
48	LDXH #nn
4A	LDXL #nn
58	LDYH #nn
5A	LDYL #nn
38	NOP
6B	OR (U) #nn
4B	OR (X) #nn
5B	OR (Y) #nn
EB	OR nnnn #nn
BB	ORA #nn
2B	ORA (U)
0B	ORA (X)
1B	ORA (Y)
AB	ORA nnnn
9E	RB #nn
93	RBC #nn
97	RBH #nn
91	RBNC #nn
95	RBNH #nn
9D	RBNV #nn
99	RBNZ #nn

Code	Mnemonic
9F	RBV #nn
9B	RBZ #nn
F1	RDA
D7	RDL (X)
D3	RDR (X)
DB	RLCA
B8	ROM1
A8	ROM2
E1	ROM2a
E3	ROM2b
D1	RRCA
8A	RTI
9A	RTS
FB	SETC
D9	SLA
D5	SRA
2E	STA (U)
0E	STA (X)
1E	STA (Y)
AE	STA nnnn
28	STA UH
2A	STA UL
08	STA XH
0A	STA XL
18	STA YH
1A	STA YL
63	STAD (U)
43	STAD (X)
53	STAD (Y)
61	STAI (U)
41	STAI (X)
51	STAI (Y)
F5	STI (X)(Y)
B1	SUBA #nn
21	SUBA (U)
01	SUBA (X)
11	SUBA (Y)
A1	SUBA nnnn
A0	SUBA UH
20	SUBA UL
80	SUBA XH
00	SUBA XL
90	SUBA YH
10	SUBA YL

take the byte from location (X) and store it into address (Y). After this, both X and Y are incremented. The flags are not affected by this operation. The other instruction, having the mnemonic CPAI (X), compares the contents of register A (the accumulator) with (X) and then increments X.

The stack instructions, having the mnemonics PSH and POP, transfer data between CPU registers and the stack, with appropriate incrementing or decrementing of the stack pointer. Flags are ignored except when POP A is executed, in which case the Z flag is affected.

Bits are shifted left or right by the SLA and SRA directives. In the case of a left shift, the most significant bit of A is stored into flag C with the least significant bit of A being cleared. A right shift produces similar results in the opposite direction. Flags Z, V and H are also affected by these operations.

Directives that shift through the carry flag are

designated by the mnemonics RLCA and RRCA. Flags Z, V and H are also affected by these instructions.

Digit rotates RDL (X) and RDR (X) rotate the hexadecimal digits of (X) through the accumulator. The instruction RDA rotates the hexadecimal digits of A, simply exchanging them. The digit rotations do not affect any flags.

The mnemonics SETC, CLRC, SETI and CLRI are used to represent the directives that set or clear the C and I flags. The mnemonic NOP signifies a "no-operation" instruction.

A jump to a specified address is produced by the instruction having the mnemonic JMP. The mnemonic JSR is used to denote a jump to a subroutine with the return address being saved in the current stack. RTS is the mnemonic used to represent the "return from subroutine" directive which removes the return address from the stack and restores it to the program counter.

Relative branches may be made in either the forward (FB) or reverse (RB) direction. An immediate data byte specifies the amount by which the program counter is advanced or decremented.

Flags may be tested to determine whether a *conditional* relative branch is to be executed. For example, FBZ specifies a forward branch only if the zero (Z) flag is set. RBNC would specify a reverse branch only if the carry (C) flag was not set.

In addition to a regular JSR directive, subroutines in the PC's BASIC ROM may be called in another manner. ROM addresses FF00 through FFFF contain a table of addresses (vectors) that are the starting addresses of subroutines that are callable through this "base page." As an example, the shortened directive CALL #5A will call the subroutine whose address is stored in ROM beginning at FF5A. An examination of the contents of ROM addresses FF5A and FF5B reveals that the actual subroutine starts at address E573.

Conditional subroutine calls, dependent on the status of a specified flag are also possible. Mnemonics for these directives have suffixes similar to those for JMP and JSR commands. CAZ (call on zero flag) and CANC (call on non-carry condition) are examples.

The last four vectors contained in the FF base page have special significance. They are the addresses, respectively, of routines for Maskable Interrupt, Timer Interrupt, Non-maskable Interrupt and Power-up procedures.

Base page vectors of even values starting with C0 and above may be accessed as one-byte machine language directives. For example, the code D8 will call the subroutine whose address is contained at FFD8 and FFD9 of the BASIC ROM's base page!

The RTI mnemonic is used to indicate a return from an interrupt routine. It differs from RTS in that, after removing a return address from the stack, it removes another byte that is stored into register E (the flags register).

BNZD represents a special instruction that makes loops easy to program. When this instruction is executed, register UL is tested, then decremented. If UL was non-zero at the time of the test, then the program counter is decremented by the number of bytes specified in the second byte of the directive. The flags are unaffected by this operation.

A 9-bit timer register may be set to produce a *timer interrupt* after a certain length of time (up to approximately .025 seconds) has elapsed. This interrupt can occur only if the I flag is set. The instructions ST0A and ST1A store 9 bits into this timer, thereby determining how long it will be before an interrupt request is made. The most significant of the nine bits will be 0 or 1 depending on which of the two instructions is used. The remaining bits are set to equal the contents of the accumulator at the time the directive is encountered.

The CPU can be halted by a WAI (wait for interrupt) directive. It will remain in that idle state until an interrupt occurs.

Finally, LDA KB loads the accumulator with a byte of input from the keyboard. DSP ON and DSP OFF turn the liquid-crystal display (LCD) on and off. The PWR OFF directive shuts down the PC.

I hope you enjoy the capabilities of the LMD!

Opcodes and Mnemonics that are Prefixed by the code FD

Code	*	Mnemonic
FD23	*	ADDA (U)
FD03	*	ADDA (X)
FD13	*	ADDA (Y)
FDA3	*	ADDA nnnn
FDEA		ADDU A
FDCA		ADDX A
FDDA		ADDY A
FD6F	*	ADNC (U) #nn
FD4F	*	ADNC (X) #nn
FD5F	*	ADNC (Y) #nn
FDEF	*	ADNC nnnn #nn
FD69	*	AND (U) #nn
FD49	*	AND (X) #nn
FD59	*	AND (Y) #nn
FDE9	*	AND nnnn #nn
FD29	*	ANDA (U)
FD09	*	ANDA (X)
FD19	*	ANDA (Y)
FDA9	*	ANDA nnnn
FD6D	*	BIT (U) #nn
FD4D	*	BIT (X) #nn
FD5D	*	BIT (Y) #nn
FDED	*	BIT nnnn #nn
FD2F	*	BITA (U)
FD0F	*	BITA (X)
FD1F	*	BITA (Y)
FDAF	*	BITA nnnn
FD8E		CLRD
FDBE		CLRI
FD27	*	CPA (U)
FD07	*	CPA (X)
FD17	*	CPA (Y)
FDA7	*	CPA nnnn
FDAC	*	DADA (U)
FD8C	*	DADA (X)
FD9C	*	DADA (Y)
FD62		DEUH
FD42		DEXH
FD52		DEYH
FD2C	*	DSBA (U)
FD0C	*	DSBA (X)
FD1C	*	DSBA (Y)
FDC0		DSP OFF
FDC1		DSP ON
FD2D	*	EORA (U)
FD0D	*	EORA (X)
FD1D	*	EORA (Y)
FDAD	*	EORA nnnn
FD60		INUH

Code	*	Mnemonic
FD40		INXH
FD50		INYH
FD25	*	LDA (U)
FD05	*	LDA (X)
FD15	*	LDA (Y)
FDAA		LDA E
FDBA		LDA KB
FDA5	*	LDA nnnn
FD58		LDX P
FD48		LDX S
FD28		LDX U
FD18		LDX Y
FD6B	*	OR (U) #nn
FD4B	*	OR (X) #nn
FD5B	*	OR (Y) #nn
FDEB	*	OR nnnn #nn
FD2B	*	ORA (U)
FD0B	*	ORA (X)
FD1B	*	ORA (Y)
FDAB	*	ORA nnnn
FD8A		POPA
FD2A		POPU
FD0A		POPX
FD1A		POPY
FDC8		PSHA
FDA8		PSHU
FD88		PSHX
FD98		PSHY
FD4C		PWR OFF
FDD7	*	RDL (X)
FDD3	*	RDR (X)
FD81		SETI
FDCE		ST0A TI
FDDE		ST1A TI
FD2E	*	STA (U)
FD0E	*	STA (X)
FD1E	*	STA (Y)
FDEC		STA E
FDAE	*	STA nnnn
FD5E		STX P
FD4E		STX S
FD6A		STX U
FD5A		STX Y
FD21	*	SUBA (U)
FD01	*	SUBA (X)
FD11	*	SUBA (Y)
FDA1	*	SUBA nnnn
FDB1		WAI

This Loader/Monitor/Disassembler was developed by:
Merlin Rober
and is published and distributed by the:
POCKET COMPUTER NEWSLETTER

P.O. Box 232, Seymour, CT 06483

 Copyright 1983 POCKET COMPUTER NEWSLETTER All Rights Reserved